DOT/FAA/AR-06/35

Air Traffic Organization
Operations Planning
Office of Aviation Research
and Development
Washington, DC 20591

Software Development Tools for Safety-Critical, Real-Time Systems Handbook

June 2007

Final Report

This document is available to the U.S. public through the National Technical Information Service (NTIS), Springfield, Virginia 22161.

U.S. Department of Transportation
Federal Aviation Administration

NOTICE

Technical Report Documentation Page

1. Report No. DOT/FAA/AR-06/35	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle SOFTWARE DEVELOPMENT TOOLS FOR SAFETY-CRITICAL, REAL-TIME SYSTEMS HANDBOOK		5. Report Date June 2007 6. Performing Organization Code
7. Author(s) Andrew J. Kornecki		8. Performing Organization Report No.
9. Performing Organization Name and Address Department of Computer and Software Engineering, College of Engineering Residential Campus Embry-Riddle Aeronautical University 600 S. Clyde Morris Boulevard Daytona Beach, FL 32114-3900		10. Work Unit No. (TRAIS) 11. Contract or Grant No. DTFA0301C00048
12. Sponsoring Agency Name and Address U.S. Department of Transportation Federal Aviation Administration Air Traffic Organization Operations Planning Office of Aviation Research and Development Washington, DC 20591		13. Type of Report and Period Covered Final Report February 2005-September 2005 14. Sponsoring Agency Code AIR-130
15. Supplementary Notes The Federal Aviation Administration Airport and Aircraft Safety R&D Division COTR was Charles Kilgore.		

16. Abstract

Since the early years of computing, programmers, system analysts, and software engineers have sought ways to improve the efficiency of the development process. Software development tools are programs that help software developers create other programs or documentation. Their objective is to automate mundane operations and bring the level of abstraction closer to the application engineer. In practice, software development tools have been in wide use among safety-critical system developers. Examples of such use, in addition to aviation, include automotive, space, nuclear, railroad, medical, and military applications. This Handbook is directed to the aviation industry and the Federal Aviation Administration to facilitate use of software development tools on airborne projects developed under DO-178B certification criteria. This Handbook outlines the issues to be considered while using development tools on software-intensive airborne systems in a regulated industry and formulates questions applicable to related DO-178B objectives. This Handbook also addresses the progress of modern software engineering and its impact on the safety-critical software development process and products.

17. Key Words DO-178B, DO-248B, Software development tools, Safety-critical systems, Tool assessment, Software development life cycle, Tool qualification	18. Distribution Statement This document is available to the public through the National Technical Information Service (NTIS) Springfield, Virginia 22161.

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	39	

Form DOT F1700.7 (8-72) Reproduction of completed page authorized

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ACRONYMS

AC	Advisory Circular
ACG	Automatic code generation
CAST	Certification Authorities Software Team
CI	Configuration index
COTS	Commercial off-the-shelf (software or hardware)
DER	Designated engineering representative
FAA	Federal Aviation Administration
HLR	High-level requirements
LLR	Low-level requirements
MBD	Model-based development
OS	Operating system
PSAC	Plan for software aspects of certification
RTCA	formerly Radio Technical Commission for Aeronautics
SAS	Software accomplishment summary
SC	Special committee
SCM	Software configuration management
SLCE	Software Life Cycle Environment
SQA	Software quality assurance
TOR	Tool Operational Requirements
TQAS	Tool qualification accomplishment summary
TQDD	Tool qualification development data
TQP	Tool qualification plan
TVR	Tool verification record

EXECUTIVE SUMMARY

Safety-critical, real-time systems, prominent in the aviation industry, continue to become more complex. They often operate in uncertain environments and must provide reliability, fault tolerance, and deterministic timing guarantees. The software for such systems may be developed using a variety of tools that must be selected to meet the needs of each specific project.

Software development tools are programs that help developers create software development artifacts (documentation, design models, source code, machine/assembly code, downloadable binary files, memory tables, etc.). Their objective is to automate mundane translation and document creation operations and bring the level of abstraction closer to the application engineer. In practice, software development tools have been in wide use among safety-critical system developers. Examples of such use, in addition to aviation, include automotive, space, nuclear, railroad, medical, and military applications.

The main objective of this Handbook and its related research project is to assess the evolving nature of software development tools for safety-critical, real-time systems and to identify how the changing nature and importance of these tools may need to be considered in the preparation of today's (and tomorrow's) guidelines for tool qualification and their use in systems certification. The Handbook organizes criteria that allow both developers and certifying authorities to evaluate specific software development tools from the system/software safety perspective with respect to their use in aviation systems undergoing the certification process. The long-term objective is to provide a basis for future software development tool qualification guidelines.

This Handbook is directed to the aviation industry and the Federal Aviation Administration to facilitate use of software development tools on airborne projects developed under DO-178B certification criteria. It outlines the issues to be considered while using development tools on software intensive systems in a regulated industry and formulates questions applicable to related DO-178B objectives. The Handbook also addresses the progress of modern software engineering and its impact on the safety-critical software development process and products.

1. INTRODUCTION.

1.1 PURPOSE AND SCOPE.

This Handbook, produced under a contract sponsored by the Federal Aviation Administration (FAA), is designed to identify assessment criteria that allow developers and certifying authorities to evaluate specific safety-critical, real-time software development tools from the system/software safety perspective.

The scope of this Handbook has been limited to software development tools that have been used, or have a potential to be used, in airborne applications. The FAA Advisory Circular (AC) 20-115B [1] introduced the principal document guiding software consideration for airborne systems, DO-178B [2], which defines a software tool as: "A computer program used to help develop, test, analyze, produce or modify another program or its documentation. Examples are an automated design tool, a compiler, test tools and modification tools." Subsequently, the document defines software development tools as: "Tools whose output is part of airborne software and thus can introduce errors." The software development tools are the focus of this research. The Handbook concentrates on development tools providing translation of design solutions into a readable version of computer code in one of the high-level language notations.

1.2 BACKGROUND.

The market of commercial software development tools is rather volatile and confusing to the developers. It often happens that, even within the same organization, one division may not be aware of the tools, either commercial off-the-shelf (COTS) or in-house developed, that are used in another division. The tools may produce artifacts in a variety of formats, requiring manual and often error-prone translation of data between the tools. The developers may face a variety of problems in an attempt to create a consistent description of the system/software properties. It should also be stressed that many general-purpose, computer-aided software engineering tools were created without understanding of or considering the DO-178B process, practically preventing tool qualification under the current guidelines. The associated research report, DOT-FAA/AR-06/36, "Assessment of Software Development Tools for Safety-Critical, Real-Time Systems," describes these issues while presenting the state-of-the-art in software development tools (as of 2003) used in safety-critical, real-time systems and providing ideas for future software development tool qualification guidelines.

1.3 AUDIENCE.

The Handbook is primarily intended for use by the Designated Engineering Representatives (DER) and Aircraft Certification Office engineers directly involved in the certification process and the certification authorities engaged in the development of policy and guidance. In addition, the Handbook will also likely be of interest to program and procurement managers, project leaders, system and software engineers, and all others directly involved in implementing software-intensive, safety-critical, real-time systems.

1.4 DOCUMENT STRUCTURE.

The Handbook consists of the following sections:

- Section 1 provides introductory material including the purpose and scope, background, audience, and directions for use.
- Section 2 includes the frame of reference based on DO-178B. It describes the key definitions and discusses development tool qualification process. It also addresses the differences between tool software and target application software.
- Section 3 presents the analysis of development tools on certified projects and discusses the tool categories and their operations.
- Section 4 focuses on the issues related to use of development tools, namely the model-driven development methodology and code generation.
- Section 5 outlines the future trends in the application of development tools.
- Section 6 is the main body of the Handbook, which includes several tables with practical comments and questions to support certification activity on projects using development tools.
- The summary in section 7 and references in section 8 close the main body of the Handbook.

1.5 USING THE HANDBOOK.

The Handbook has been designed to help the aviation industry and certifying authorities handle situations when in-house developed or COTS software development tools are used on a safety-critical, real-time system developed under DO-178B guidelines. The starting point is to review sections 2 to 5 of the Handbook to rediscover the issues related to the software development tools and their qualification, and to review the tool use from the broader modern software development perspective. Section 6 provides a practical collection of issues and questions to be raised when developing/reviewing a project which uses software development tools.

Note: This Handbook is the result of and complements the related research effort. It does not, in and of itself, constitute policy or guidance. The FAA may use this Handbook in the creation of future policy or guidance.

2. DO-178B FRAMEWORK.

In 1980, the Radio Technical Commission for Aeronautics, now RTCA, convened a special committee (SC-145) to establish guidelines for developing airborne systems and equipment software. They produced a report: "Software Considerations in Airborne Systems and Equipment Certification," which was subsequently approved by the RTCA Executive Committee and published in January 1982 as RTCA document DO-178. After gaining further experience in airborne software certification, the RTCA decided to revise the previously published document. Another committee (SC-152) drafted DO-178A, which was published in 1985.

Due to rapid advances in technology, the RTCA established a new committee (SC-167) in 1989. Its goal was to update, as needed, DO-178A. SC-167 focused on five major areas:

- Documentation integration and production
- System issues
- Software development
- Software verification
- Software configuration management (SCM) and software quality assurance (SQA)

The resulting document, DO-178B, provides guidelines for these areas. Also, a key addition to this updated version was the concept of tool qualification.

Subsequently, two other documents were created that have a critical bearing on the subject. RTCA DO-248B [3] clarifies some of the material in DO-178B. FAA Order 8110.49 [4] compiles a variety of guidelines related to the use of software in airborne systems. Specifically, chapter 9 is dedicated to tool qualification.

2.1 DEFINITIONS AND INTERPRETATIONS.

The interpretation of the term tool qualification might vary from one organization to another. According to the definition given in DO-178B:

> "Tool Qualification - The process necessary to obtain certification credit for a software tool within the context of a specific airborne system."

While:

> "Certification credit - Acceptance by the certification authority that a process, product or demonstration satisfies a certification requirement."

Explanation of the purpose and the need for tool qualification (DO-178B, Section 12.2):

> "The objective of the Tool Qualification is to ensure that the tool provides confidence at least equivalent to that of the process(es) eliminated, reduced, or automated."

> "A tool may be qualified only for use on a specific system …Use of the tool for other systems may need further qualification."

> "Only those functions that are used to eliminate, reduce, or automate software life cycle process activities, and whose outputs are not verified, need be qualified."

Tool Qualification is a supplementary process the applicant may elect to follow in a course of certification for the airborne system. It is the certification authority that decides the outcome of the qualification process. Moreover, qualification, if claimed, is considered as a requirement to get the system certified. There is a significant amount of work involved to qualify a development tool. Note that numerous development tools have been used successfully in certification projects without being qualified.

One often-repeated statement regarding development tool qualification is the requirement that "only deterministic tools can be qualified." The DO-178B refers to tool determinism as "… tools which produce the same output for the same input data when operating in the same environment." This definition does not take into account how the output is generated. One may interpret that it is not required to provide proof on the internal behavior of the tool. An example of this can be found in a tool running on a host workstation in a multitasking, multiuser, networked environment. The problem is one of defining the object code for the tool. Does it include the operating system (OS) of the host workstation? A tool clearly needs to make explicit calls to the OS routines, and any verification of these would require full visibility of the host OS and related high assurance of its operation.

A recommendation on this subject could be: If the usage of OS routines is necessary for a tool, such routines should be identified and verified. The tool software structural coverage analysis at source level must include the coverage of these calls. Any activity on the object code level would include the impact of compiler and demonstration of traceability to the object code level for multitasking. Pipelined architecture with multilevel processor cache may be too difficult to verify.

DO-248B section 3.61 introduces the question: "What constitutes a development tool and when should it be qualified?" The interpretation, based on section 12.2 of DO-178B, provides the following answer: "Examples of tools are: compilers, automated code generators, linkers, GUI builders, automated database construction, and graphical modeling tools that generate source code." Such tools all have something in common: they take one artifact as input and produce another as an output. They are basically translators.

The problems with development tool qualification typically originate from the fact that the modern tool—typically a complex software development environment—has the translation component hidden deeply within the tool. These tools usually provide a variety of other functions, which are not directly related to the translation process. This is why the qualification, if applicable, should be focusing on this translation component of the tool functionality. In practice, the actual translation algorithm is considered intellectual property and rarely can be disclosed. There is no data available on the requirements, design, or code for this internal tool feature. Unless the tool is of an in-house variety, or the details of its design can be disclosed, the qualification efforts are most likely doomed.

2.2 NEED FOR QUALIFICATION.

Typical use of a software development tool—called a software producer—is to transform an input artifact into an output, thus creating another software artifact. Is there a need to qualify the translating software for every step in the lifecycle? If this transformation has an impact on the final airborne product, the producer needs to be qualified, but only if the transformation output would not be verified. The current process mandates verification after each transformation.

Hence, there are no incentives to expend a significant effort on qualification of the development tool when a much smaller effort on output verification leads to the same outcome.

FAA Order 8110.49 describes conditions when a tool requires qualification:

> "There are three questions to ask to determine if a tool needs qualification. If the answer is "Yes" to all of the questions below, the tool should be qualified:
>
> (a) Can the tool insert an error into the airborne software or fail to detect an existing error in the software within the scope of its intended usage?
>
> (b) Will the tool's output not be verified as specified in Section 6 of DO-178B?
>
> (c) Are processes of DO-178B eliminated, reduced, or automated by the use of the tool? That is, will the output from the tool be used to either meet an objective or replace an objective of DO-178B, Annex A?"

The business case might be made for qualification if it would be possible to qualify a tool one time, to be used on multiple projects. However, the current language of DO-248B states:

> "The certification authority considers the software as part of the airborne system or equipment installed on the certified aircraft or engine; that is, the certification authority does not approve the software as a unique, stand-alone product."

This is interpreted that no software, and by extension no software development tool, could be certified or qualified by itself. Software must be associated with a specific certified airborne system. Contrarily, FAA Order 8110.49, chapter 12 and AC 20-148 [5] on reusable components promote the need for reusability of software components, as well as tools. However, the current practices of not packaging tool data separately, lack of comprehensive tool lifecycle documentation, and close coupling of the tool and the application make reusability much less feasible.

2.3 QUALIFICATION PROCESS.

Tool qualification is permitted only for a tool used as part of a specific certification project, e.g., part of a Type Certificate, Supplemental Type Certificate, or Technical Standard Order approval. The tool data are referenced within the Plan for Software Aspects of Certification and Software Accomplishment Summary (SAS) documents for the original certification project. The applicant should make the Tool Operational Requirements (TOR) available for review. The TOR describes tool functionality, environment, installation, operation manual, development process, and expected responses (also those in abnormal conditions). Two tool-specific documents must be submitted: Tool Qualification Plan (TQP) and Tool Qualification Accomplishment Summary (TQAS) (FAA Order 8110.49, chapter 9). To support qualification, the applicant must demonstrate that the tool complies with its TOR under both normal and abnormal operating conditions. This demonstration may involve a trial period during which a verification of the tool output is performed and tool-related problems are analyzed, recorded, and corrected. The document also states that software development tools should be verified to check the correctness, consistency, and completeness of the TOR and to verify the tool against those requirements. More data are required for the qualification of a development tool, including tool configuration management index, TQAS, tool development data, tool verification data, tool quality assurance records, tool configuration management records, etc. These requirements are also described in chapter 9 of the FAA Order 8110.49. The tool qualification development data (TQDD) are approved only in the context of the overall software development for the specific system where the intention to use the tool is stated in the PSAC. The tool itself does not receive a separate qualification stamp of approval. Use of the tool for other systems may need a separate qualification, although some qualification credits may be reused.

The following steps are required:

a. The developer creates and submits to the Certification Authority the PSAC document including specific reference to the TQP document.

b. The developer must specify as part of airborne product PSAC the intent to use the development tool with references to the tool data, TQP, and baseline qualification approach.

c. The TOR document, which includes references to the qualification tests conducted to prove that the tool operates correctly and reliably in the development environment, is made available to the Certification Authority.

d. To complete the certification process, the SAS document references the TQAS, which in turn is based on the tool verification record (TVR) and the TQDD with the tool-related data including tool software design, code, test cases and procedures, and the references to the activities for evaluating the qualification variables on the avionic hardware and the software platforms.

e. The qualification process is complete when the submitted TQP and TQAS documents are approved as evidence that the tool complies with the TOR under normal and abnormal operation conditions. The TOR, TVR, and TQDD documents must be available for review. Additional documents, such as tool version description, configuration index (CI), requirements document, verification procedures and results, may be also required.

2.4 DEVELOPMENT TOOL SOFTWARE VERSUS TARGET SOFTWARE.

The main difference between tool software and embedded target software is that the tool does not execute on embedded hardware. Typically, the tool operates in a general-purpose workstation environment. The tool is not embedded, and instead, it runs typically under conventional COTS operating system. In fact, the development tool may be considered a ground-based system, and the critical consideration for the tool is the integrity and correctness of the generated artifact. In this respect, the tool software is similar to software used in ground-based systems.

In the case of embedded software, if untested (and faulty) code is executed, the behavior of the software could be erroneous, which could impact the system behavior and ultimately system safety. A consequence of an unintended activation of untested code for a tool may have a safety consequence only when erroneous code would be generated for future use on the target. To avoid this impact, one of the development tool requirements must be to generate code only in normal situations.

By definition, the tool's output has an impact on the target software. Thus, it is imperative to ensure the correctness of a tool's output. Typically, there are no constraints related to timing as long as the correct output will be produced. However, for example, the development tool may exhibit errors and even crash during development due to OS hang-up or an outside factor (e.g., network traffic, virus). The code produced before the crash can still be correct for use in the airborne system. Current DO-178B wording does not seem to consider the distinction between errors exhibited in the development environment and errors exhibited in the target airborne environment. Future guidelines should consider these characteristics using approaches elaborated in DO-278 [6], which addresses nonairborne Communications, Navigation, Surveillance, and Air Traffic Management systems and in DO-200A [7], which describes processing of aeronautical data.

The objectives of the development tool's software verification process are different from those of the verification process for target software. The tool's high-level requirements (HLR) correspond to the TOR rather than the requirements of the target system. Verification of software development tools may be achieved by (a) review of the TOR and demonstration that the tool complies with its TOR under both normal and abnormal operating conditions (in the latter case it is not producing output which could be used in the target software), (b) requirements-based testing and structural coverage analysis, as appropriate, and (c) analysis of potential errors produced by the tool.

According to the collected industry feedback, the following DO-178B requirement for the development tool is overly restrictive:

> "If a software development tool is to be qualified, the software development processes for the tool should satisfy the same objectives as the software development processes of airborne software."

This requirement implies that the qualification process for a development tool is the DO-178B process itself. The existing wording of DO-178B (section 12.2) and its further elaboration in FAA Order 8110.49 (chapter 9) states that for a software development tool to be qualified, the software development processes for the tool should satisfy the same objectives as the software development processes of the target airborne software. To soften this restriction, the guidelines allow applicants reduction of the software level, e.g., from level A to B. The document postulates that such reduction must be justified, while leaving wide open the interpretation on how such justification is to be made.

3. USE OF TOOLS ON CERTIFIED PROJECTS.

Due to various issues, including intellectual property and competitive advantage, there is no central repository that maintains records of previously qualified tools. Only the applicant, who qualified a tool within the scope of a specific certification project, retains the necessary data. The research team made attempts to identify software development tools that have been qualified and found only a handful. In addition, the obtained information was anecdotal based on personal contacts and word of mouth rather than documented in a way that the research team could examine in detail.

The research discovered that industry uses numerous simple tools of rather limited functionality, which are developed in-house and which often are considered as an integral component of the applicant project in terms of the certification activities. Such tools typically (a) support a simple function like translating software artifact from one format to another, (b) are executed from a command line or have a very simple interface, and (c) their documentation is not publicly available. Due to intellectual property constraints and business practices, it is rather difficult to obtain specific information about such tools.

Commercial tools are typically multifunctional development environments with very sophisticated graphical interfaces. In the limited instances when a commercial tool has been qualified, the actual qualification was limited to single functionality of the tool (e.g., code generator). Certainly, the vendor and the applicant had to build a close relationship that allowed applicants detailed access to tool intellectual property and cooperation in supporting claims for satisfying the DO-178B objectives. Generic data about tools are abundant; however, the access to the qualification data is again restricted.

3.1 TYPES OF DEVELOPMENT TOOLS.

Modern tools are rather complex and may be categorized in a variety of ways. Since these categories may overlap, a single tool may belong to a number of categories. The following list identifies tool categories by functionality and provides examples of specific functionalities that they provide:

- The requirements category includes tools used early in the life cycle to identify and specify the software requirements. Also included are the tools that help determine the correctness of the requirements using semiformal models, even though some could argue that such tools belong more to the analysis category (functionality examples: definition, specification, interface analysis, requirements formal modeling, properties verification, traceability, version management, etc.).

- The analysis category includes tools used for analysis of software behavior and timing, typically before and after the product is developed. Since aviation software is always developed within the system context, analysis tools typically deal with the system requirements and performance (functionality examples: throughput, timing, sizing, simulation, etc.).

- The testing category spans over the entire life cycle, since the testing must deal with all software artifacts (functionality examples: coverage, test case generation, etc.).

- The design category includes tools that support executable models and are used for requirements verification, design implementation, code generation, and the development of documentation and test cases (functionality examples: modeling the system using applicable graphical notation in form of blocks, objects, diagrams, code generation, documentation, reuse, etc.).

- The implementation category includes all support required to take the computer code and transfer it to the executable program (functionality examples: Integrated Development Environment: editor, compiler, debugger, linker, loader, browser, target customization, etc.).

- The target category includes application run-time support software components that are not considered to be development tools in the sense of the DO-178B definition, since they are clearly components of the target system (functionality examples: real-time OS board support package, libraries, etc.).

Tools that can be classified into the above design, implementation, or target categories support the DO-178B coding and integration phases. Many modern design tools generate code as well as test cases. They frequently support executable models with options to verify various requirements properties. It often happens that a system already produced may be reused or reengineered, thus many tools provide a vehicle for architectural or design pattern use. The boundary between the tool categories is frequently fuzzy. The growing tendency is to work at higher levels of abstraction using design or analysis models as the source code relying on automatic code generation (ACG) and compilers to produce target software. Model-based development (MBD) has become the Lingua Franca of modern software engineering [8 and 9].

3.2 CONTROL VERSUS SOFTWARE PARADIGM.

During the design of a real-time system, it is important to be aware that there are two distinct classes of modern systems exposed to environmental stimuli [10]:

- Interactive—the computer system determines the pace of operation by granting or allocating resources to clients on request when feasible (OSs and data bases). The concerns are deadlock avoidance, fairness, and data coherence.

- Reactive—the system environment determines the pace of operation while the computer system reacts to external stimuli producing outputs in a timely way (process control, avionics, and signal processing). The concerns are correctness and timeliness.

Software engineers are very familiar with the concepts of OSs, programming languages, software development methodologies, and notations. The proliferation of object-oriented methodologies is replacing previous procedural approaches. The graphic notations supported by these techniques allow developers to represent the software of the target system as a set of components that are easy to translate into programming constructs (modules, objects, methods, functions, procedures, and data structures) using the ACG functionality of the tool. The developer needs to have full understanding of the generated target code and often needs to fill in the framework generated by the tool with specific code in the target language.

However, control engineers consider the system as a dynamic model consisting of well-defined blocks of specific functionality (logic, arithmetic, and dynamic). The data-flow paradigm of the model supports its simulation and analysis of behavior. Subsequently, the model can be translated automatically into an equivalent code, typically without any additional developer involvement.

Software engineers, concentrating on computer operations, are more accustomed to interactive systems, while control (or system) engineers, who are educated in control theory, focus on reactive systems. This may be called software engineering and control engineering paradigms, respectively. Unfortunately, most complex safety-critical, real-time systems include characteristics of both paradigms. There is no unified theory to represent both paradigms in a smooth way. This dualism is reflected in the variety of modern software development tools, which attempt to bridge the gap. The challenge of the contemporary tool market is to cater to the software and control engineers with the tool providing appropriate support for both paradigms. The gap between software and control professionals makes communication of critical design decisions difficult and may be one of the causes of misunderstandings that are unacceptable for safety-critical system design.

Categorization of tools used for modeling the system reflects these diverse viewpoints of safety-critical, real-time systems developers resulting from their different backgrounds. Two viewpoints are exemplified in software development tools, which use either a function-based, block-oriented approach, or a structure-based, object-oriented approach.

Both categories include a graphical user interface that allows the user to specify a design in a graphical or textual manner and a set of functionalities to save, load, modify, and execute (or animate) such representation. Both typically include target code generation capability and, thus, they are not treated as separate categories.

3.2.1 Structural-Based Software Design and Modeling Tools.

Structural-based software design and modeling tools, preferred by engineers with software and computer background, contain all commonly used design tools based on structural decomposition. The tools from this category are based on object-oriented decomposition and unified model language-like modeling of the system, allowing the software developers to create a model describing both the structure and behavior. The structural-based tools are software development oriented and match the interactive paradigm.

3.2.2 Functional-Based Software Design and Modeling Tools.

Functional-based software design and modeling tools, preferred by engineers with system and control background, contain all commonly used design tools based on functional decomposition. These tools allow the domain specialist (e.g., control engineer) to build a model describing the system functionality represented as block diagrams with their input-output transformations. The tools in this category are more system and control than software development oriented and conform to the reactive paradigm.

3.3 DEVELOPMENT TOOL OPERATIONS.

DO-178B section 12.2 requires the development of a TOR for all tools to be qualified. For a development tool, the TOR is a detailed requirements document that is traced to the design, code, and test cases of the tool itself. Despite several attempts to obtain such data from industry, such data was not available for this research. In the place of an actual TOR, four basic functionality items for a design tool with ACG capability have been identified. The tool allows the developer to

- create design by using notation supporting specific development methodology.
- verify correctness of the design.
- confirm the dynamic behavior of the design.
- generate code.

This is intended to be a generalized list of functions that can apply to several tools. Each item can be further analyzed in detail to create a refined list of the functionalities, which in turn can be assessed in practical evaluation experiments.

For the source code generated by a tool, one still needs a compiler to create the executable code and a linker to link it with other executable objects. However, the compilers are usually not qualified. To gain assurance and demonstrate that the compiler would not introduce errors in the embedded software, the following characteristics must be incorporated, at minimum:

- The generated code should be very simple using a limited number of specific language constructs, yielding a linear code in a form of a sequence of macro calls (procedure or function).

- The compiler is used on a hand-coded software subset and fully tested for complete coverage analysis.

- The compiler must be used in the same configuration, options, and environment as the one used to compile the remaining hand-created objects.

4. ISSUES.

Qualification of development tools, even when proper guidelines are provided, is not an often sought option in the airborne software industry. In fact, one could argue that qualification of development tools is not a viable option. The current interpretation of guidelines makes development tools qualification difficult from the technical viewpoint (if even possible) and impractical from a managerial or cost viewpoint. The development tool needs to be qualified to the same level of scrutiny as the appropriate application it is helping to develop—several DO-178B objectives are not applicable to the tool software and can not be met. The intellectual property of the specific development tool may need to be disclosed by the vendor to achieve qualification. The tools that could be considered for qualification are usually very simple, typically in-house created utilities, where the applicant holds all intellectual property, has all tool development data, and can reuse the tool software artifacts in consecutive projects. The simplicity of the in-house tools, based on a simple automatic transformation, allows the applicant to perform nearly exhaustive testing and show the tool determinism. Modern COTS tools are generally very large, complex, multifunctional tools that do not come with adequate tool data to facilitate tool qualification.

The tool cannot be qualified as a stand-alone software artifact. The qualification is accomplished within the scope of a specific certification project and, thus, is not clearly visible from the outside as development tool qualification. The current proliferation of MBD and the wide use of design tools with code generation capability make these tools prime candidates for qualification.

There seems to be a lack of consensus on the issue of what is to be considered source code in the modern MBD-driven paradigm. What is considered as determinism when related to the development tool is also open for discussion. However, there does seem to be an agreement that the development tools must be very cautiously tested and verified before their output can be trusted. The simplicity of tool function, separation of concerns, partitioning, and use of model checking and formal evaluation are the leading factors to consider on how the development tools could meet the safety needs.

The research showed that some interest exists to qualify software development tools classified in the functional-based, block-oriented control engineering paradigm category. However, despite the wide use of the software engineering paradigm, there was little to no interest to qualify the tools classified in the structural-based, object-oriented category, although these tools have been widely used to create software for safety-related systems over a range of industries. According to several informal exchanges with industry, most commercially available control engineering tools have been used in creation of software artifacts on certified projects. The research received anecdotal information about few in-house tools qualified on various projects. Several tools were developed internally and received early recognition with the user community, but failed to make it in the commercial market.

The software industry is a volatile industry with companies both growing fast and declining fast. Software products become obsolete due to frequent modification of the computer hardware and OS platforms. For example, a tool working in a DOS environment may not be appropriate in a Microsoft® Windows® environment. Software developers are changing their company affiliations and taking the intelligence that is necessary to maintain and upgrade the tools and development environments with them. Software products (including software development tools) are overtaken by another company, e.g., after a merger or buyout, and software products are reissued under different names with different logos and slightly modified sales pitches.

Overall—despite arguments to the contrary from a small group of developers, DER, and tool vendors—the research indicates that, taking into account the current guidelines, the industry is not considering the development tool qualification as a priority issue.

The days of handcrafting the entirety of source code seems to be nearing an end. As described, regardless of the development approach, a design tool needs to have ACG capability in order to compete on the market. The quality of the transformation is critical for overall assessment of software development tool quality. Features such as determinism, correctness, robustness, and conformance to standards will be considered as the assessment characteristics. The tool's extent in ensuring traceability between artifacts generated from one development phase to another can be a starting point to make the arguments about tool quality [11].

In fact, code generators have become modern versions of compilers. As the compiler translates high-level language source code into executable machine code, the code generator translates a graphical model (or, to be more specific, its internal representation) into high-level language code. The level of abstraction is now raised to the point where the system and software architects and designers can contribute more to the engineering solution.

It is interesting to note that the issue of compiler acceptability may be associated with other development tools, particularly those with code generation capability. DO-178B section 4.4.2 reads: "Upon successful completion of verification of the software product, the compiler is considered acceptable for that product." DO-248B Section 3.31 provides an interpretation of this statement:

> "The language and compiler selected need to support the achievement of the verification objectives. Verification planning needs to take into account compiler features, as they have impact on the verification process. The compiler is considered acceptable once all of the verification objectives are satisfied, but the compiler is only acceptable for that product and not necessarily for other products. This does not qualify the compiler as a development tool."

The code generation process feature of the tool must be analyzed under both normal and abnormal operating conditions to determine likely causes of unintentional or erroneous code generation. One conventional system hazard identification and assessment technique may be used to identify hazardous failures. Identified failures need to be assessed for criticality, and possible mitigating measures need to be proposed. Examples of failures may include:

- OS failure during code generation
- Improper code generator initialization and/or selection of parameters (code generation options)
- Inconsistency between the generated software and the target hardware (due to incorrect generation, tool deficiency, and/or selection of options/parameters mismatched with the target)
- Problems related to memory overflow resulting in faulty code generated
- Incorrect code generation algorithm inserting or omitting the data (commission or omission errors)

The code generation feature of a tool must also be analyzed to show direct mapping of the artifacts entered by the developer conforming to the specific tool modeling paradigm and the components of the resulting generated code. For many modern tools, which provide incredibly complex and multifaceted views of the system, there may be a need to select only a subset of the available modeling components to assure the easy mapping with the generated code and its subsequent verification. It is necessary to analyze the limits and constraints in terms of the number of components, connections, hierarchy levels, etc.

The design tools discussed in this research would allow the user to create graphical representations of the software behavior that the tool would convert into modules of computer code. An early approach was to create specific library of macroassembly modules for each graphical functional block that could be placed on a diagram. Due to its simplicity, a framework program that instantiates the macros and stitches together the inputs to the outputs of the different blocks could be evaluated for correctness using the actual version of the certification guidance. The assembly language representation was reasonably close to the machine execution representation that the correctness could be reviewed. This introduced almost no uncertainty in the correctness of the tool output, thus facilitating the qualification of several simple in-house tools based on this principle. However, today's complex approach for tool implementation almost precludes such ease of review.

5. FUTURE TRENDS.

The obvious question is: Why were there only a few attempts to qualify development tools? The research shows that more verification tools are qualified than development tools. This may be due to less stringent criteria for qualifying verification tools than for development tools. Another issue is that of economics. In the aviation industry, methodologies and design approaches typically do not last longer than two airplane programs. Therefore, making a large investment in a tool cannot be spread over a large number of programs to get a good return on investment. Some companies began development tool qualification, but stopped their efforts after realizing that under current guidelines and regulations it is not justifiable from a cost perspective.

The industry and certifying authorities are actively engaging in discussions on the topic of development tool qualification. Several software tool vendors are working with avionics developers, certification authorities, and DERs to identify approaches to practically address development tool qualification. The international Certification Authorities Software Team (CAST) has documented several related position papers (available on the FAA website at http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/). The CAST position papers were coordinated among the software specialists of certification authorities from the United States, Europe, and Canada. However, they do not constitute official policy or guidance from any of the authorities. These documents are provided for educational and informational purposes only and should be discussed with the appropriate certification authority when considering use for actual projects. Specifically, CAST Paper 13 [12] reflects observations closely related to those described in this report.

> "More precisely, the primary issue for an ACG [Automatic Code Generation] tool is the production of source code that does not comply with its requirements, but can still be compiled without any error detected and is executable. For airborne software, the same event can occur, but it's not the only one. Halts during

execution, overflows, variations in time response, hardware and software incompatibilities, hardware failures, unbounded recursive algorithms, bad stack usage, resource contention, tasks conflicts, bad interaction with others systems, etc., are examples of issues which may jeopardize flight safety, if they appear in aviation software. However, these types of errors may not have any influence on the flight safety, if they occurred in the ACG tool that generated the aviation software."

And subsequently:

> "The applicant should demonstrate that the tool is designed and developed in such a way that erroneous functioning of the operating system cannot produce unintended or erroneous code (e.g., showing tool operational requirements in abnormal conditions). Neither can it jeopardize determinism properties of the tool (i.e., the produced code should not differ when the input data does not, as previously mentioned)."

Using service history as an alternate qualification method is addressed both in DO-178B section 12.3.5 and DO-248B DP 11 and is also referenced in section 9-6.i of FAA Order 8110.49. This method can be used only for software that has been used for an extended time without being previously qualified. These original documents are not clear what the applicant has to supply to get that type of qualification. CAST Paper 1 [13] elaborates on product service history attributes such as duration, amount and quality of data, number of errors, number of modifications, change control, and contributing to the product acceptability and, thus, to obtaining certain certification and qualification credit. New research on service history [14] provides additional guidance on this topic. However, implementation of such an approach is still a matter of interpretation and must be agreed upon between the applicant and the certifying authority on a case-by-case basis. CAST Paper 22 [15] explores potential reuse of software components to tool qualification data. The rapid evolution of tools, the long duration of projects, and the requirement to requalify tool versions make the use of service history difficult.

Another consideration for development tool qualification is the increasing interaction between airborne and ground-based systems due to the data up and downlink capability. The increasing tendency of using COTS systems is noticeable in military, medical, transportation, and space applications [16 and 17]. The similarities are visible since the tool's operating environment is typically COTS and certainly the tool output has an impact on target software operation.

6. COMMENTS AND QUESTIONS ABOUT DEVELOPMENT TOOLS.

This section is designed to provide practical guidance for applicants and certification personnel when dealing with application of software development tools on certification projects.

6.1 QUALIFICATION CRITERIA.

Table 1 presents criteria applicable to development tool qualification. It also includes comments reflecting the research results based on analysis of relevant documentation and feedback received from industry.

Table 1. Criteria Applicable to Development Tool Qualification

DO-178B Reference	DO-178B Criteria	Development Tool Handbook Comment
12.2	Only deterministic tools may be qualified.	There are wide and narrow interpretations of determinism. For a wide interpretation, an argument has been made that determinism holds for a tool that may produce different outputs for the same input providing all the outputs can be shown to be correct. For a narrow interpretation, the output must be unique. The combined definition would be that a tool is deterministic if its output reflects correctly and uniquely a defined input sequence. An argument about determinism should be a critical part of a tool's qualification package.
12.2	Qualification should be only for a specific system; the intention should be stated in the PSAC.	The industry would like to see tools qualified for easy reuse on different projects. The term qualifiable has been used to describe tools that have been qualified earlier and whose qualification documents could be made available. This goal is attainable if the tool qualification package includes specific descriptions and limitations of the tool operational environment and qualification is limited only to selected feature(s) of the tool with precise description of the intended functionality and the produced output. Such a package could be reused on different projects providing the conditions are not changed.
12.2 b	Combined tools should be qualified to comply with paragraph DO-178B 12.2.1 unless partitioning can be shown.	For a combined development and verification of a tool, it is critical to separate and partition tool feature(s) related to the development activity. Qualification will address this (these) feature(s) only.
12.2 c	SCM and SQA process objectives should be applied to tools being qualified.	With the rapid progress of technology, most hardware platforms and software versions are upgraded in nearly monthly cycles. It is of utmost importance to keep track of the tool operational environment and version control. Only rigorous SCM and SQA can provide arguments for tool qualification. On a practical side, many software tool vendors are not familiar with the airborne software guidelines, and they lack proper mechanisms to maintain the configuration data supporting their claims.

Table 1. Criteria Applicable to Development Tool Qualification (Continued)

DO-178B Reference	DO-178B Criteria	Development Tool Handbook Comment
12.2.1.a	Qualification should satisfy the same objectives as the airborne software.	This is a difficult and restrictive requirement for tool qualification. A software tool works in a different environment than the target software and has different operational requirements. The tool may crash due to interference from the underlining operating system (e.g., Microsoft Windows) but which may not impact the produced target code. For tool qualification, several objectives can potentially be eliminated.
12.2.1.b	The software level of the tool may be reduced.	In reference to 12.2.1.a, this item should be applied with less restriction.
12.2.1.d (1)	TOR should be reviewed.	The tool for developing the target software is used under a limited set of operational scenarios defining the constraints of the operational requirements. The operational requirements will define the specific tool functionalities, which are qualifiable tool features.
12.2.1.d (2)	Compliance with TOR under normal operating conditions should be demonstrated.	This is the critical element in tool qualification activity.
12.2.1.d (3)	Compliance with TOR under abnormal operating conditions should be demonstrated.	The potential hazards should be identified to determine likely causes of unintentional or erroneous tool outputs. A standard hazard identification and assessment technique may be used to identify hazardous failures, which need to be assessed for criticality and possible mitigating measures.
12.2.1.d (4)	Requirements-based coverage should be analyzed.	This may be one of the most challenging items to comply with since both COTS vendors and in-house software developers are reluctant to provide detailed and specific development tool requirements documentation (due to the intellectual property issues, business drivers, lack of availability, etc.).
12.2.1.d (5)	Structural coverage appropriate for the tool's software level should be completed.	The access to the development tool source code is limited to very narrow group of the original developers. Opening such to external scrutiny could be too much of a challenge.
12.2.1.d (6)	Robustness testing appropriate for the tool's software level should be completed.	Robustness and testing the consistency of development tool operation, the ability to evolve, and fault tolerance should be fundamental criteria for the tool evaluation. It needs to be noted that the robustness for the tool is different than robustness for the target software, see 12.2.1.a.
12.2.1.d (7)	Potential errors should be analyzed.	The tool hazard analysis should be a base for these analyses, see 12.2.1.d (3).

Based on FAA Order 8110.49, figure 9-1.

6.2 QUALIFICATION DATA.

Table 2 presents the data required for the development tool qualification by FAA Order 8110.49 (figure 9.2). It shows that there are two formal documents to be submitted by applicants to support the qualification claim: TQP and TQAS. These documents must be referenced in the applicant certification package documents: PSAC and SAS, respectively. In addition, for the development tools TOR, TVR, and TQDD must be available for review. All required documents must be complete and distinctively related to the environment and configuration of the specific application.

Table 2. Data Required for Development Tool Qualification

Data/Document	Available/Submit	DO-178B Reference
PSAC referencing TQP of the tool to be qualified	Submit	12.2, 12.2.3.a, and 12.2.4
TQP	Submit	12.2.3.a(1), 12.2.3.1, and 12.2.4
TOR	Available	12.2.3.c(2) and 12.2.3.2
SAS referencing TQAS of the tool to be qualified	Submit	12.2.4
TQAS	Submit	12.2.3.c(3) and 12.2.4
TVR e.g., test cases, procedures used to test the tool with their results	Available	12.2.3
TQDD e.g., tool requirements, design, and code	Available	12.2.3

6.3 TOOL OPERATIONAL REQUIREMENTS QUESTIONS.

TOR are critical to provide information about what a tool does, how it is to be used, and its operational environment. Table 3 lists several questions addressing the guidelines for evaluating acceptability of TOR as described in section 9.6 of FAA Order 8110.49.

Table 3. Evaluating Acceptability of TOR

FAA Order 8110.49 Reference	TOR-Related Questions for Development Tools
9.6 a (1)	Do the tool qualification tests allow verification of the tool functionality in terms of requirements as specified in the TOR?
9.6 a (2)	Is the operational environment of the tool specified in the TOR? Is there a clear and complete description of the OS, version, hardware, and interfaces? Is there a clear description of tool limitations in terms of what the tool will not do?
9.6 a (3)	Is the tools user's manual available and included in the TOR package? Does the user's manual match the used version of the tool?
9.6 b (1)	Does TOR document include a clear description of the software development process performed by the tool?
9.6 b (2)	Are abnormal operating conditions specified? Are the tool responses under abnormal conditions tested?
9.6 c	Does the TOR identify the tool features that are directly related to the activity for which the qualification is being sought? Is there an argument presented that the features not directly related to the qualification activity have no adverse effect on the features used?
9.6 c	Is it possible to provide arguments that the tool output produced as a result of a defined input sequence is correct and reflects this input? Note: Even though there could be two or more different outputs produced, it needs to be shown that all outputs are correct.

6.4 GENERAL DO-178B QUESTIONS.

DO-178B includes a dedicated Section 12.2, dealing exclusively with tools. This section was later clarified in chapter 9 of FAA Order 8110.49. The previous tables 1, 2, and 3 presented the compilation of development tool issues extracted from these documents. However, in addition to section 12.2, other sections of DO-178B make multiple references to the use of tools. Table 4 identifies these references and proposes additional questions related to the use of development tools to be asked by the certification authorities.

Table 4. DO-178B Statements Related to the Development Tools

DO-178B Reference	DO-178B Statement	Related Questions for Development Tools
2.1.2	Information Flow from Software Processes to System Processes The system safety assessment process determines the impact of the software design and implementation on system safety using information provided by the software life cycle processes. This information includes fault containment boundaries, software requirements, software architecture, and error sources that may have been detected or eliminated through software architecture or by the use of tools or by other methods used in the software design process. Traceability between system requirements and software design data is important to the system safety assessment process.	Does the software development tool provide means to trace the design elements to the system safety requirements? Does the tool provide means to provide information on software requirements, architecture, fault containment boundaries, and error sources? Are there data available supporting claims of traceability between the artifacts on the tool input and output?
4.1	Software Planning Process Objectives c. The software life cycle environment, including the methods and tools to be used for the activities of each software life cycle process have been selected (subsection 4.4).	Are all development tools to be used on the project identified in the PSAC and software planning documents? See 4.3.b (3)
4.2	Software Planning Process Activities c. Methods and c. tools should be chosen that provide error prevention in the software development processes. f. When multiple-version dissimilar software is used in a system, the software planning process should choose the methods and tools to achieve the error avoidance or detection necessary to satisfy the system safety objectives. i. If user-modifiable code is planned, the process, tools, environment, and data items substantiating the guidelines of paragraph 5.2.3 should be specified in the software plans and standards.	Are there any data available and referenced in software plan to support claims that the development tool does not introduce errors?
4.3	Software Plans b. The software plans should define the criteria for transition between software life cycle processes by specifying: (3) Availability of tools, methods, plans and procedures.	Do software plans address development tools in terms of availability, support, maintenance, and training?

Table 4. DO-178B Statements Related to the Development Tools (Continued)

DO-178B Reference	DO-178B Statement	Related Questions for Development Tools
4.4	Software Life Cycle Environment Planning The purpose of the planning for the software life cycle environment is to define the methods, tools, procedures, programming languages and hardware that will be used to develop, verify, control and produce the software life cycle data (section 11) and software product. The basic principle is to choose requirements development and design methods, tools, and programming languages that limit the opportunity for introducing errors, and verification methods that ensure that errors introduced are detected. The considerations presented above may affect: • The software development environment tools.	Is there a long enough track record for using the development tool to justify claims that errors introduced by the tool are limited? Are the means of verification of the development tool output identified and explained?
4.4.1	Software Development Environment Guidance for the selection of software development environment methods and tools includes: b. The use of qualified tools or combinations of tools and parts of the software development environment should be chosen to achieve the necessary level of confidence that an error introduced by one part would be detected by another. An acceptable environment is produced when both parts are consistently used together. d. If certification credit is sought for use of the tools in combination, the sequence of operation of the tools should be specified in the appropriate plan. e. If optional features of software development tools are chosen for use in a project, the effects of the options should be examined and specified in the appropriate plan. Note: *This is especially important where the tool directly generates part of the software product. In this context, compilers are probably the most important tools to consider.*	Is there data available to show that the intermediate artifacts created by a tool are correct before they are made available to the subsequent life cycle phase? If the tool generates an artifact constituting part of the target software product, have the options for generating this artifact been recorded and used consistently?

Table 4. DO-178B Statements Related to the Development Tools (Continued)

DO-178B Reference	DO-178B Statement	Related Questions for Development Tools
7.2.9	Software Life Cycle Environment Control The objective of SLCE is to ensure that the tools used to produce the software are identified, controlled, and retrievable. The software life cycle environment tools are defined by the software planning process and identified in the Software Life Cycle Environment Configuration Index (subsection 11.15). Guidance includes: a. Configuration identification should be established for the Executable Object Code (or equivalent) of the tools used to develop, control, build, verify, and load the software. b. The SCM process for controlling qualified tools, should comply with the objectives associated with Control Category 1 or 2 data (subsection 7.3), as specified in paragraph 12.2.3, item b. c. Unless 7.2.9 item b applies, the SCM process for controlling the Executable Object Code (or equivalent) of tools used to build and load the software (for example, compilers, assemblers, and linkage editors) should comply with the objectives associated with Control Category 2 data, as a minimum.	Is the development tool included in the SLCE CI. Is the executable code of the development tool under configuration control? If the development tool is qualified, does it comply with SCM Control Category 1?
11.1	Plan for Software Aspects of Certification g. Additional considerations: This section describes specific features that may affect the certification process, for example, alternative methods of compliance, tool qualification, previously developed software, option-selectable software, user-modifiable software, COTS software, field-loadable software, multiple-version dissimilar software, and product service history.	Are the tools to be qualified identified in the project PSAC and, if a development tool(s) is to be qualified, are the qualification means described in the TQP and is the TQP referenced in the PSAC?

Table 4. DO-178B Statements Related to the Development Tools (Continued)

DO-178B Reference	DO-178B Statement	Related Questions for Development Tools
11.2	<u>Software Development Plan</u> c. <u>Software development environment</u>: A statement of the chosen software development environment in terms of hardware and software, including: (1) The chosen requirements development method(s) and tools to be used. (2) The chosen design method(s) and tools to be used. (3) The programming language(s), coding tools, compilers, linkage editors, and loaders to be used. (4) The hardware platforms for the tools to be used.	Are all development tools used on the project described in the software development plan? Does the description include the development tools operational environment and the hardware platforms?
11.4	<u>Software Configuration Management Plan</u> (a) <u>Environment</u>: A description of the SCM environment to be used, including procedures, tools, methods, standards, organizational responsibilities, and interfaces. (b) <u>Activities</u>: A description of the SCM process activities in the software life cycle that will satisfy the objectives for: (9) <u>Software life cycle environment controls</u>: Controls for the tools used to develop, build, verify and load the software. This includes control of tools to be qualified.	Does the SCM Plan include description of the tool(s) with respect to their operating environment and the means of control of this environment?
11.5	a. <u>Environment</u>: A description of the SQA environment, including scope, organizational responsibilities and interfaces, standards, procedures, tools and methods.	Does the SQAP include a description of SQA environment as related to the development tool(s) used on the project?

Table 4. DO-178B Statements Related to the Development Tools (Continued)

DO-178B Reference	DO-178B Statement	Related Questions for Development Tools
11.6	<u>Software Requirements Standards</u> The purpose of Software Requirements Standards is to define the methods, rules, and tools to be used to develop the high-level requirements. These standards should include: a. The methods to be used for developing software requirements, such as structured methods. b. Notations to be used to express requirements, such as data flow diagrams and formal specification languages. c. Constraints on the use of the requirement development tools. d. The method to be used to provide derived requirements to the system process.	Does the Software Requirements Standard define constraints on tools used for development of the requirements?
11.7	<u>Software Design Standards</u> The purpose of Software Design Standards is to define the methods, rules, and tools to be used to develop the software architecture and low-level requirements. These standards should include: a. Design description method(s) to be used. b. Naming conventions to be used. c. Conditions imposed on permitted design methods, for example, scheduling, the use of interrupts and event-driven architectures, dynamic tasking, re-entry, global data, exception handling, and rationale for their use. d. Constraints on the use of the design tools. e. Complexity restrictions, for example, exclusion of recursion, dynamic objects, data aliases, and compacted expressions.	Does the Software Design Standard define constraints on tools used for development of the software architecture and low-level requirements? Does the tool documentation explicitly list the modeling constructs and operations that the developers can use? Are the limitations (i.e., constructs and operations that cannot be used) identified?

Table 4. DO-178B Statements Related to the Development Tools (Continued)

DO-178B Reference	DO-178B Statement	Related Questions for Development Tools
11.8	Software Code Standards The purpose of Software Code Standards is to define the programming languages, methods, rules and tools to be used to code the software. These standards should include: e. Constraints on the use of the coding tools.	Does the Software Code Standard define constraints on tools used for development of the code? Does the tool documentation list the command sequences and operations that can be used to generate the code? Are the parameters, configuration data, etc., defined and kept under version control?
11.15	Software Life Cycle Environment Configuration Index The Software Life Cycle Environment Configuration Index (SECI) identifies the configuration of the software life cycle environment. This index is written to aid reproduction of the hardware and software life cycle environment, for software regeneration, reverification, or software modification, and should: a. Identify the software life cycle environment hardware and its operating system software. b. Identify the software development tools, such as compilers, linkage editors and loaders, and data integrity tools (such as tools that calculate and embed checksums or cyclical redundancy checks). c. Identify the test environment used to verify the software product, for example, the software verification tools. d. Identify qualified tools and their associated tool qualification data. Note: *This data may be included in the Software Configuration Index.*	Does the SLCE CI identify the configuration of development tools used on the project? Does the tool documentation explicitly define the tool environment in terms of the operating system and version used, file structure, environmental variables, paths, configuration files, options, data, etc.? Does the SLCE CI identify the qualified tools and their qualification data?

Table 4. DO-178B Statements Related to the Development Tools (Continued)

DO-178B Reference	DO-178B Statement	Related Questions for Development Tools
12.1.3	Change of Application or Development Environment Use and modification of previously developed software may involve a new development environment, a new target processor or other hardware, or integration with other software than that used for the original application. New development environments may increase or reduce some activities within the software life cycle. New application environments may require activities in addition to software life cycle process activities which address modifications. Guidance for change of application or development environment includes: a. If a new development environment uses software development tools, the guidelines of subsection 12.2, Tool Qualification, may be applicable.	If a development tool is used in a new development environment, does the tool comply with the certification objectives? Are there data to support claims for meeting the objectives in case of using a new hardware or operating system version for the development tool? Are there data to support claims for meeting the objectives in case of the tool integration with other software than that used for the original application?
12.3	Alternative Methods Alternative Methods may be used to support one another. For example, formal methods may assist tool qualification or a qualified tool may assist the use of formal methods.	Are there data to support justification of alternative methods to be used for tool qualification? Are the data describing clearly and completely the approach and supporting evidence?
12.3.3.4	Tool Qualification for Multiple-Version Dissimilar Software: If multiple-version dissimilar software is used, the tool qualification process may be modified, if evidence is available that the multiple software development tools are dissimilar. This depends on the demonstration of equivalent software verification process activity in the development of the multiple software versions using dissimilar software development tools. The applicant should show that: a. Each tool was obtained from a different developer. b. Each tool has a dissimilar design.	Are there data to show that development tools used for multiple version were obtained from different sources and they have different designs? Can it be demonstrated for dissimilar software that equivalent verification activity was carried out for each version?

Section 12.2 is not included in this table.
SQAP = Software Quality Assurance Plan
SLCE = Software Life Cycle Environment
CI = Configuration index

SQA = Software Quality Assurance
SDP = Software Development Plan
SCM = Software Configuration Management

6.5 TOOL EVALUATION: CONCERNS AND QUESTIONS.

Airborne software, as any other software installed in safety-critical, real-time systems, must meet high-level criteria of dependability and safety assurance. For software development tools from the qualification perspective, the research identified five critical concerns to be addressed when evaluating the tool for use on certification projects:

- Determinism
- Robustness
- Traceability
- Correctness
- Conformance to Standards

Table 5 presents these concerns with their related DO-178B references. The table includes potential questions related to the objectives the tool needs to achieve to address these concerns. The table may be used to support qualification activity in a situation when there is full access to tool software artifacts and the tool was developed in conformance with DO-178B guidelines.

Table 5. Software Development Tool Qualification—Evaluation Matrix

Concerns	Subcategory	Software Development Tool Question	DO-178B Reference
Determinism	Source code	Does analysis of the tool code structure provide coverage appropriate to the assurance level?	6.4.4.2a 6.4.4.2b
		Does analysis of the tool code structure confirm data and control coupling appropriate to the assurance level?	6.4.4.2c
Robustness	HLR	Is the tool executable object code robust with HLR?	6.4.2.2 6.4.3
	LLR	Is the tool executable object code robust with LLR?	6.4.2.2 6.4.3
	Source code	Is the software development tool partitioning integrity confirmed?	6.3.3f
Traceability	SR to HLR	Are the tool HLR traceable to the TOR?	6.3.1f
	HLR to LLR	Are the tool LLR traceable to the tool HLR?	6.3.2f
	LLR to code	Is tool source code traceable to the tool LLR?	6.3.4e
	HLR to test	Is test coverage of the tool HLR achieved?	6.4.4.1

Table 5. Software Development Tool Qualification—Evaluation Matrix (Continued)

Concerns	Subcategory	Software Development Tool Question	DO-178B Reference
	LLR to test	Is test coverage of the tool LLR achieved?	6.4.4.1
Correctness	HLR	Are the tool HLR accurate and consistent?	6.3.1b
	LLR	Are the tool LLR accurate and consistent?	6.3.2b
	Source code	Is the tool source code accurate and consistent?	6.3.4f
Conformance to Standards	HLR	Do the tool HLR conform to standards?	6.3.1e
	LLR	Do the tool LLR conform to standards?	6.3.2e
	Source code	Does the tool source code conform to standards?	6.3.4d

SR = System requirements HLR = High-level requirements LLR = Low-level requirements

The related research on assessment of software development tools for safety-critical, real-time systems explored literature on software quality [18, 19, and 20] and software evaluation [21, 22, 23, and 24]. The investigation identified several useful questions for the development team related to the tool adaptation, from both the perspectives of the manager and the developer. They address the issues of resource adaptation, tool reputation, vendor support, usability, self-documentation, teamwork support, analysis capability, and safety and conformance. They are not included in this Handbook. Any interested reader may refer to section 4 of the associated report DOT/FAA/AR-06/36, "Assessment of Software Development Tools for Safety-Critical, Real-Time Systems."

7. SUMMARY.

The use of software development tools is a reality for all facets of software development. Use of such tools in the highly regulated aviation industry is more cautious, and justifiably so. When a development tool is used on a certification project, a large number of questions must be asked and adequate responses received to assure the certifying authorities that the tool performs as intended. Typically, verification activities perform this task. Alternatively, software development tools may be qualified although, currently, they must satisfy the same criteria as the airborne software in which they are used and may not be qualified as a stand-alone product.

To encourage feasibility of a development tool's stand-alone qualification, concepts such as component-based software, software reuse, and service history may be explored. For complex multifunctional tools, partitioning of the separate functionalities must be clearly defined and the specific internal tool software data must be available. The issues of software development tool version control and precise definition of operational environment, constraints, and limitations are the basis for starting discussion about tool qualification. The availability of extensive tool software development data, often scarce for COTS products, may be a challenge to accomplishing COTS tools qualification (although re-engineering may help).

The issues identified in this Handbook are regulatory and managerial in nature. The major hurdle is the current state of regulations and guidelines. The secondary obstacle is the business model and lack of incentives, i.e., prohibitive costs of tool qualification. The existing tools, often used in certification projects, lack appropriate data to be used as artifacts in meeting the objectives of DO-178B. The applicant's objective is system certification rather than tool qualification. The system integrator is concerned with the target software and not with the intermediate tool-supported transformations. A tool seller may not see a business advantage in tool qualification since that may require disclosing proprietary information to potential competitors, and a tool developer may not be familiar with the requirements and the rigors of airborne software. In reality, development tool qualification typically requires close collaboration of all interested parties: application developer, system developer, system integrator, the tool vendor (potentially including an access to the actual tool software developer's team), and the applicant.

Internal trade studies[1] have shown that, due to the more stringent assurance requirements, the cost of the development tool qualification is higher than the cost of verification tool qualification. The use of qualified verification tools can result in fast savings on the first program where they are introduced. In contrast, the use of qualified development tools may require several programs to make up the cost.

A second group of issues is related to the state of software evaluation. Currently, there is no agreement on what metrics would allow developers to carry an independent and unbiased tool assessment. It could be conceivable to create an independent laboratory dedicated to tool qualification and encourage commercial tool vendors to submit their product for assessment. Similar approaches are already operational in the general area of verification and validation. Another idea would be to require certified product applicants to disclose information regarding the development tool use and qualification effort by creating an FAA-sponsored database for DO-178B certified products. This could be met with serious objections from industry due to their apprehensiveness to disclose information that might cause the loss of a commercial advantage. Another recommendation is to research the potential for development tool qualification using an approach different than one outlined in section 12.2 of DO-178B. In addition to service history, other options may include formal methods [25, 26, and 27], dependability assessment [28], usage based assessment [29], etc. Another option would be to consider development tools as ground-based software. Such an approach would allow focusing on the tool's software integrity and using guidelines in DO-278 and DO-200A. All of these may contribute to a possible update of DO-178B considering the recent rapid progress in software engineering as a discipline.

For a comprehensive solution, a new approach may be needed that would take into account some new developments and facts in software engineering and safety-critical systems. For example,

[1] Software development tools are claimed to cost 20 times more to qualify as verification tools (internal data from Honeywell, ERAU/FAA Software Tool Forum, Bill Potter presentation, slide 13).

several vendors have announced new products based on the MBD principle, supporting the development of graphical or textual models and subsequent ACG. Most of these products also have means of model execution and verification. Such solutions move the process to where it is most needed: the front end of the development life cycle.

Another issue is: how to handle tool evolution? Tools, as any software product, have a tendency to evolve and change rapidly. Often, the development cycle of certification projects lasts much longer than the lifespan of the tools used on the project. It forces developers to use earlier versions of the tool while a new one would be available. In some cases, vendors are going out of business or merging with others, and suddenly a tool re-emerges under a new name with a slightly modified interface and functionality. The problem is that the original documentation may not be maintained to provide a mechanism that prevents the tool from exhibiting some idiosyncrasies or hidden features known only to the original tool developers. An upgrade indicated by a new version to a previously qualified tool or the operating environment would be a reason for re-qualification.

Research indicates that the pressing need of industry is to identify methods to qualify a tool that is independent of a specific program and applications using it. This would require updating the guidelines to consider the model-driven development paradigm, redefining the qualification process, and allowing flexibility regarding qualification that is less dependent on the application program using the tool. A more streamlined method to qualify development tools and to keep them current as technology advances would be immensely useful. Better guidance on how to apply service history, handle COTS tools, reuse qualification data, and how to address what has to be done for incremental tool changes is also needed.

8. REFERENCES.

Almost 150 references were examined for this project. The complete list is in section 8 of the related report DOT/FAA/AR-06/36, “Assessment of Software Development Tools for Safety-Critical, Real-Time Systems.” The following is a subset selected to present the positions directly related to the issues presented in this Handbook. They are representative of existing regulations, guidelines, and trends in software engineering related to the software development tool assessment and use.

1. U.S. Department of Transportation, Federal Aviation Administration, AC 20-115B, “Advisory Circular – Subject: RTCA Inc., Document RTCA/DO-178B,” January 1993.

2. RTCA DO-178B, “Software Considerations in Airborne Systems and Equipment Certification,” Radio Technical Commission for Aeronautics, RTCA SC-167, December 1992.

3. RTCA DO-248B, “Final Annual Report for Clarification of DO-178B ‘Software Considerations in Airborne Systems and Equipment Certification,’” Radio Technical Commission For Aeronautics, RTCA SC-190, October 2001.

4. U.S. Department of Transportation, Federal Aviation Administration, FAA Order 8110.49, "Software Approval Guidelines," FAA, 2003 (Chapter 9 replaces FAA Notice N8110.91 of 2001).

5. U.S. Department of Transportation, Federal Aviation Administration, AC 20-148, "Advisory Circular: Reusable Software Components," December 2004.

6. RTCA DO-278, "Guidelines for Communications, Navigation, Surveillance, and Air Traffic Management (CNS/ATM) Systems Software Integrity Assurance," Radio Technical Commission for Aeronautics, RTCA SC-190, 2002.

7. RTCA DO-200A, "Standards for Processing Aeronautical Data," Radio Technical Commission for Aeronautics, RTCA SC-181, 1998.

8. Dickens, M., Sharp, J., and Ledin, J., "Integrated Modeling Environment Constructs High-Fidelity Plant and Controller Models," RTC Magazine, March 2004.

9. Bichler, L., Rademacher, A., and Schurr, A., "Evaluating UML Extensions for Modeling Real-Time Systems," Proceedings of 6th International Workshop on Object-oriented Real-time Dependable Systems, WORDS2001, Roma, Italy, January 8-10, 2001.

10. Berry, G., *The Foundation of Esterel*: *Essays in Honour of Robin Milner*, MIT Press, 1998.

11. Whalen, M.W. and Heimdahl, M.P.E., "An Approach to Automatic Code Generation for Safety Critical Systems," *Proceedings of the 14th IEEE International Conference on Automated Software Engineering*, Orlando, October 1999.

12. Certification Authorities Software Team (CAST) Paper 13, "Automatic Code Generation Tools Development Assurance," June 2002.

13. Certification Authorities Software Team (CAST) Paper 1, "Guidance for Assessing the Software Aspects of Product Service History of Airborne Systems and Equipment," June 1998.

14. Ferrel U.D. and Ferrel, T.K., "Software Service History Report," FAA report DOT/FAA/AR-01/125, January 2002.

15. "Reuse of Software Tool Qualification Data Across Company Boundaries (Applying the Reusable Software Component Concept to Tools," Certification Authorities Software Team (CAST) Paper 22, March 2005.

16. Freeland, G.I., "COTS Tools Reduce DO-178B Software Development Impact," *COTS Journal*, February 2002, pg. 29-35.

17. Krodel, J., "Commercial Off-The-Shelf Avionics Software Study," FAA report DOT/FAA/AR-01/26, May 2001.

18. “Information Technology–Software Product Evaluation–Quality Characteristics and Guidelines for Their Use,” ISO/IEC 9126, Geneva, Switzerland, 1991.

19. Barbacci, M., Klein, M.H., Longstaff, T.A., and Weinstock, C.B., “Quality Attributes,” Software Engineering Institute, Technical Report, CMU/SEI-95-TR-021, Pittsburgh, PA, December 1995.

20. VanSuetandael, N. and Elwell, D., “Software Quality Metrics,” FAA report DOT/FAA/CT-91/1, August 1991.

21. “Information Technology – Guideline for the Evaluation and Selection of CASE Tools,” IEEE Std. 1462-1998, IEEE Standards Board, March 1998.

22. Firth, R., Mosley, V., Pethia, R., Gold, R., and Wood, W., “A Guide to the Classification and Assessment of Software Engineering Tools,” SEI, Technical Report CMU/SEI-87-TR-10, ESD-TR-87-111, August 1987.

23. Wichmann, B., “Guidance for the Adoption of Tools for Use in Safety Related Software Development,” British Computer Society, March 1999.

24. Cornella-Dorda, S., Dean, J., Lewis, G., Morris, E., Oberndorf, P., and Harper, E., “A Process for COTS Software Product Evaluation,” Technical Report, Software Engineering Institute, CMU/SEI-2003-TR-017, 2003.

25. Rushby, J., “Formal Methods and Their Role in the Certification of Critical Systems,” SRI International Company, August 1995.

26. McDermid, J.A., “Software Safety: Where’s the Evidence?”, *Proceedings of the 6th Australian Workshop on Industrial Experience with Safety-Critical Software*, Brisbane, Australia, 2001.

27. Lawlis, P., Mark, K., Thomas, D., and Courtheyn, T., “A Formal Process for Evaluating COTS Software Products,” *Computer Magazine*, May 2001, pp. 58-63.

28. Fenton, N.E., Littlewood, B., Neil, M., Strigini, L., Sutcliffe, A., and Wright, D., “Assessing Dependability of Safety-Critical Systems Using Diverse Evidence,” *IEEE Proceedings Software Engineering*, 145(1), Limerick, Ireland, 2000, pp. 35-39.

29. Voas, J., “Developing a Usage Based Software Certification Process,” *Computer Magazine*, August 2000, v.33, n.8, pp. 32-37.